Towards an Extended Model of User Interface Adaptation: The ISATINE Framework

Víctor López-Jaquero[1], Jean Vanderdonckt[2], Francisco Montero[1], and Pascual González[1]

[1] Laboratory on User Interaction & Software Engineering (LoUISE)
Universidad de Castilla-La Mancha, 02071 Albacete, Spain
{victor,fmontero,pgonzalez}@dsi.uclm.es
[2] Belgian Laboratory of Computer-Human Interaction (BCHI)
Université catholique de Louvain, 1348 Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

Abstract. In order to cover the complete process of user interface adaptation, this paper extends Dieterich's taxonomy of user interface adaptation by specializing Norman's theory of action into the ISATINE framework. This framework decomposes user interface adaptation into seven stages of adaptation: goals for adaptation, initiative, specification, application, transition, interpretation, and evaluation. The purpose of each stage is defined and could be ensured respectively by the user, the interactive system, a third party, or any combination of these entities. The potential collaboration between these entities suggests defining additional support operations such as negotiation, transfer, and delegation. The variation and the complexity of adaptation configurations induced by the framework invited us to introduce a multi-agent adaptation engine, whose each agent is responsible for achieving one stage at a time (preferably) or a combination of them (in practice). In this engine, the adaptation rules are explicitly encoded in a knowledge base, from which they can be retrieved on demand and executed. In particular, the application of adaptation rules is ensured by examining the definition of each adaptation rule and by interpreting them at run-time, based on a graph transformation system. The motivations for this multi-agent system are explained and the implementation of the engine is described in these terms. In order to demonstrate that this multi-agent architecture allows an easy reconfigurability of the interactive system to accommodate the various adaptations defined in the framework, a case study of a second-hand car-selling system is detailed from a simple adaptation to progressively more complex ones.

Keywords: Adaptation, adaptation configuration, delegation, isatin, ISATINE framework, mixed-initiative user interface, multi-agent system, negotiation, reconfiguration of user interface, transfer, user interface description language.

1 Introduction

We are witnessing a paradigm shift in the interaction with computers. The progressive migration of applications from desktop PCs to mobile platforms is changing the habits of user in interaction. Furthermore, a new mass of computer interaction neophytes is

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 374–392, 2008.

becoming attracted to the possibilities of using computer applications to support many daily tasks, such us buying flight or theater tickets. At the same time, as communications and hardware sensors cost gets cheaper the availability of information to the applications is quickly increasing. To take advantage of this increase in the information available to the application from the context of use where they are executed, adaptation mechanisms that adjust the application according to the data received from the context of use need to be devised. For this purpose, a multitude of adaptation techniques are been used [3,11,14].

Currently, the most widely accepted understanding of the adaptation process comes from Dieterich's survey of adaptation techniques [5], despite that it has been produced in 1993. In addition to its age, Dieterich's taxonomy suffers from several shortcomings: it is constrained by only entities (e.g., the user and the system) in each stage of the adaptation process, it does not handle an explicit collaboration and it is restricted to the execution only. Furthermore, some of the most relevant issues in the adaptation process such as how the adaptation is specified were left out of the framework. In particular, Dieterich's taxonomy is incomplete with respect to the seven stages of Norman's theory of action [14]. This model describes how a user interacts with an application from the beginning, when the user is forming his intention to reach a goal, until the end, when the user evaluates the results from the actions taken to achieve the goal.

This paper expands Dieterich's framework by incorporating some extra stages adapted from the mental model proposed by Norman. These extra stages improve user involvement in adaptation process and foster a more detailed description of how the adaptation process is carried out. To validate the proposed framework, an architecture supporting the framework is also presented. The architecture has been designed as a multi-agent system to enable easy extensibility and to make more natural the design of the negotiation, transferring and delegation capabilities required for the adaptation stages proposed in our framework to be executed collaboratively.

This paper starts by describing the ISATINE adaptation framework (Section 2), along with the antecedents that have motivated and inspired it. Section 3 introduces a multi-agent architecture that supports the proposed framework and describes how each adaptation stage proposed in the framework is supported. A discussion on how the designed multi-agent architecture has been implemented is delivered in Section 4. Section 5 exemplifies the framework by applying the framework and the architecture on a running example: a second-hand car selling application with various levels of adaptation. Some conclusions and future work are reported in Section 6.

2 The ISATINE Framework for User Interface Adaptation

This section first introduces Dieterich's taxonomy of user adaptation in order to identify its shortcomings, thus initiating an extension according to Norman's theory of action for user interaction [14] resulting in the ISATINE framework. This framework is defined in the second subsection.

2.1 Dieterich's Taxonomy of User Adaptations

On the one hand, Dieterich's taxonomy of user adaptations has always been considered as a seminal reference for classifying different types of user interface adaptation

configurations and techniques. This paper sorted more than 200 papers dealing with various forms of user interface adaptation and summarized them into four stages needed to perform any form of adaptation, in principle:

1. **Initiative:** one of the entities involved in the interaction suggests its intention to perform an adaptation. The main entities are usually the user and the system.
2. **Proposal:** if a need for adaptation arises, it is necessary to make proposals of adaptations that could be applied successfully in the current context of use for that need for adaptation detected.
3. **Decision:** as we may have different proposals from the previous stage, which adaptation proposal best fit the need for adaptation detected should be decided, and whether it is worth applying each proposal.
4. **Execution:** finally, the adaptation proposal chosen will be executed. One important factor when making any changes in the UI is how the transition from the original UI to the adapted one is performed. Before the execution stage, a prologue can be executed to prepare the UI for the adaptation. For instance, if the adaptation includes switching from one code to another code, the prologue function should store the current state of the application, so it can be resumed after the adaptation takes places. On the other hand, an epilogue function can be provided to restore the system after adaptation takes place. This epilogue will take care of restoring application state and resuming the execution of the application.

On the other hand, we are considering Norman's mental model of user interaction which decomposes any user interaction into seven Stages of Action:

1. Forming the Goal: the user shapes a goal in her mind.
2. Forming the Intention: to reach the goal, the user is forming some intention.
3. Specifying an Action: the intention is turned into a series of actions.
4. Executing an Action: one action at a time is selected and executed.
5. Perceiving the State of the World: after that the action has been executed, the results produced by this action are perceived.
6. Interpreting the State of the World: the results perceived trigger an interpretation in the user's mind on how the World has changed.
7. Evaluating the Outcome: depending on this interpretation, the user evaluates whether the action she executed matches her initial goal or not.

If we attempt to match Dieterich's four stages of adaptation on Norman's model, it can be observed straightforwardly that the initiative corresponds to the intention, that the proposal and the decision are two steps involved in the action specification, and that both execution stages match (Fig. 1). Therefore, only some portion of the whole process, the left part of Norman's model, is covered, thus creating a need for covering the remaining uncovered portion. This expresses some current shortcomings such as: the results of adaptation should be made perceivable in a way that is appropriate enough for the user to understand it. Not only the adaptation results could be made perceivable, but also the adaptation execution itself. Too often interactive systems supporting some adaptation do not convey properly the idea and the meaning of the adaptation process. Empirical studies have shown that users are always confused to some extent when they face some adaptation. If nothing is implemented in the system to minimize this effect, the adaptation process is likely to be rejected. Fig. 1 does not reveal when the adaptation is performed by the user (adaptable user interface) vs. by

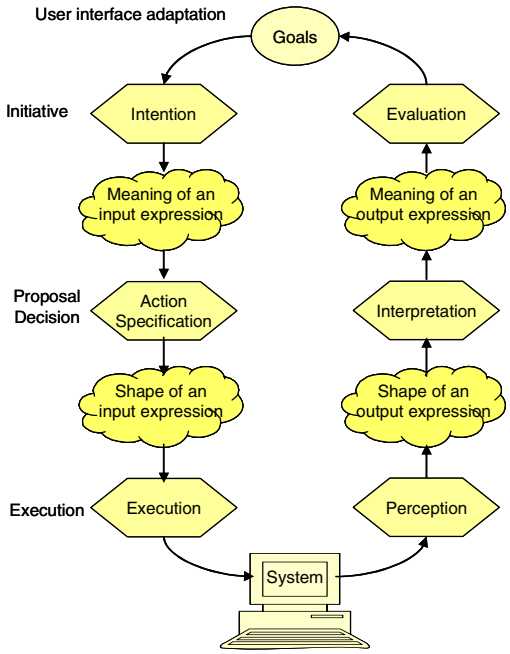

Fig. 1. The four steps of Dieterich's taxonomy located on Norman's mental model

the system (adaptive user interface). In Norman's model, goals are typically expressed by a human trying to interact with the system. Therefore, there is a need to better identify the roles of each entity. Dieterich's model does not decompose very much the adaptation process into sub-processes, thus leaving some room for more expressivity.

2.2 Definition of the ISATINE Framework

The shortcomings identified in the previous subsection lead us to expand Dieterich's taxonomy by trying to express the adaptation process according to all the Seven Stages of Norman's model. In this way, it is expected that no adaptation stage will be left out. Basically, we state that three entities are involved in the adaptation process: the *user* (U), the *interactive system* (S), or any *third party* (T), which may substitute the two previous entities in case of need (e.g., request for help, further support, support for some operation which is impossible to achieve otherwise, failure). When at least two entities share the responsibility of a stage, there is a need for coordinating the input and output of these entities. For instance, mixed-initiative [9] represents a typical configuration when U and S collaborate to determine the best option possible for ensuring a stage. We distinguish three forms of coordination:

1. *Negotiation*: options could be presented by each entity and the final result is negotiated between the entities so as to reach a consensus. T could serve for this purpose when, for instance, contradicting output are produced by U and T. Or for stating which entity has the higher priority.

2. *Delegation*: when an entity estimates that it does not have information or responsibility enough to achieve the adaptation stage, it may request help/support from any other entity to achieve its purpose. When the results come back to the requesting entity, it may then decide the final option, therefore keeping the control over the decision process.
3. *Transfer*: this form is the same as delegation, but without any return to the requester. The requested entity takes the decision and may send a notification.

The specialization of Norman's model for adaptation results into the ISATINE[1] framework, so-called because the Seven Stages became seven adaptation stages, each one being specialized for each entity (Fig. 2):

1. *Goals for user interface adaptation*: any entity (U, S, or T) may be responsible for establishing and maintaining up-to-date a series of goals to ensure user interface adaptation. Although this adaptation is always for the final benefit of the user, it could be achieved with respect to any aspect of the context of use (with respect to the user herself, the computing platform used by the user, or the complete physical and organizational environment in which the user is carrying out her task). The goals are said to be *self-expressed*, *machine-expressed*, *locally* or *remotely*, depending on their location: in the user's head (U), in the local system (S), or in a remote system (T). A typical example of machine-expressed goals is encountered when the system is made responsible for maintaining a certain level of fault-tolerance depending on varying network or hardware conditions. This main goal could be further decomposed into sub-goals, like keeping a minimal amount of information, ensuring a graceful degradation [7] of the user interface, or avoiding any task disruption.
2. ***I****nitiative for adaptation*: this stage is further refined into formulation for an adaptation request, detection of an adaptation need, and notification for an adaptation request, depending on their location: respectively, U, S, or T. For example, T could be responsible for initiating an adaptation when an update of the UI is made available or there is a change of context that cannot be detected by the system itself (e.g., an external event).
3. ***S****pecification of adaptation*: this stage is further refined in specification by demonstration, by computation, or by definition, depending on their origin: respectively, U, S, or T. When the user wants to adapt the UI, she should be able to specify the actions required to make this adaptation, such as by programming by demonstration or by designating the adaptation operations required. When the system is responsible for this stage, it should be able to compute one or several adaptation proposals depending on the context information available. When the third party specifies the adaptation, a simple definition of these operations could be sent to the interactive system so as to execute them.

[1] An orange-red crystalline substance, $C_8H_5NO_2$, obtained by the oxidation of indigo blue. It is also produced from certain derivatives of benzoic acid, and is one important source of artificial indigo (Source: http://dictionary.reference.com/).

4. ***A****pplication of adaptation*: this stage specifies which entity will apply the adaptation specified in the previous stage. Since this adaptation is always applied on the UI, this UI should always provide some mechanism to support it. If U applies the adaptation (e.g., through UI options, customization, personalization), it should be still possible to do it through some UI mechanisms.
5. ***T****ransition with adaptation*: this stage specifies which entity will ensure a smooth transition between the UI before and after adaptation. For instance, if S is responsible for this stage, it could provide some visualization techniques, which will visualize the steps, executed for the transition, e.g., through animation, morphing, progressive rendering [15].
6. ***IN****terpretation of adaptation*: this stage specifies which entity will produce meaningful information in order to facilitate the understanding of the adaptation by other entities. Typically, when S performs some adaptation without explanation, U does not necessarily understand why this type of adaptation has been performed. Conversely, when U performs some adaptation, she should tell the system how to interpret this evaluation. For instance, [6] develops a machine-learning algorithm where the system first proposes some adaptation to be applied. If this adaptation does not correspond to users' needs, the user provides the alternate adaptation instead and tells the system how to incorporate this new adaptation scheme for the future. The system updates the knowledge base by interpreting this explanation.
7. ***E****valuation of adaptation*: this stage specifies the entity responsible for evaluating the quality of the adaptation performed so that it will be possible to check whether or not the goals initially specified are met. For instance, if S maintained some internal plan of goals, it should be able to update this plan according to the adaptations applied so far. If the goals are in the users' mind, they could be also evaluated with respect to what has been conducted in the previous stages. In this case, the explanation of the adaptation conducted also contributes to the goals update. Collaboration between S and U could be also imagined for this purpose.

The only stage, which could not be a priori ensured by U or T, is the execution, unless the user is a programmer or the third part supports dynamic programming.

The deviation between the initial expression of goals for UI adaptation and those specified in terms of the system is referred to as the *adaptation semantic distance in input*. When an adaptation operation is adequately specified, the deviation between this specification and the operations required to achieve the adaptation step is referred to as the *adaptation articulatory distance in input*. The sum of these two deviations denotes the gulf of adaptation execution and represents how complex it could be to represent and execute the adaptation operations in the system's terms. Similarly, the deviation between the perception of the adaptation as performed on the UI and the perception of the user denotes the *adaptation articulatory distance in output*. The difference between the goals reached so far in the system and the initial goal denotes the *adaptation semantic distance in output*. The sum of these two deviations denotes the gulf of adaptation evaluation and represents how complex it could be to evaluate the results of the adaptation. This second gulf is too often forgotten in adaptation algorithms, although it is largely reported (e.g., in [3,5]) that any adaptation, however good and adequate it could be, always provokes some perturbation in the user's mind. By reducing this gulf, the perturbation should be able to be minimized.

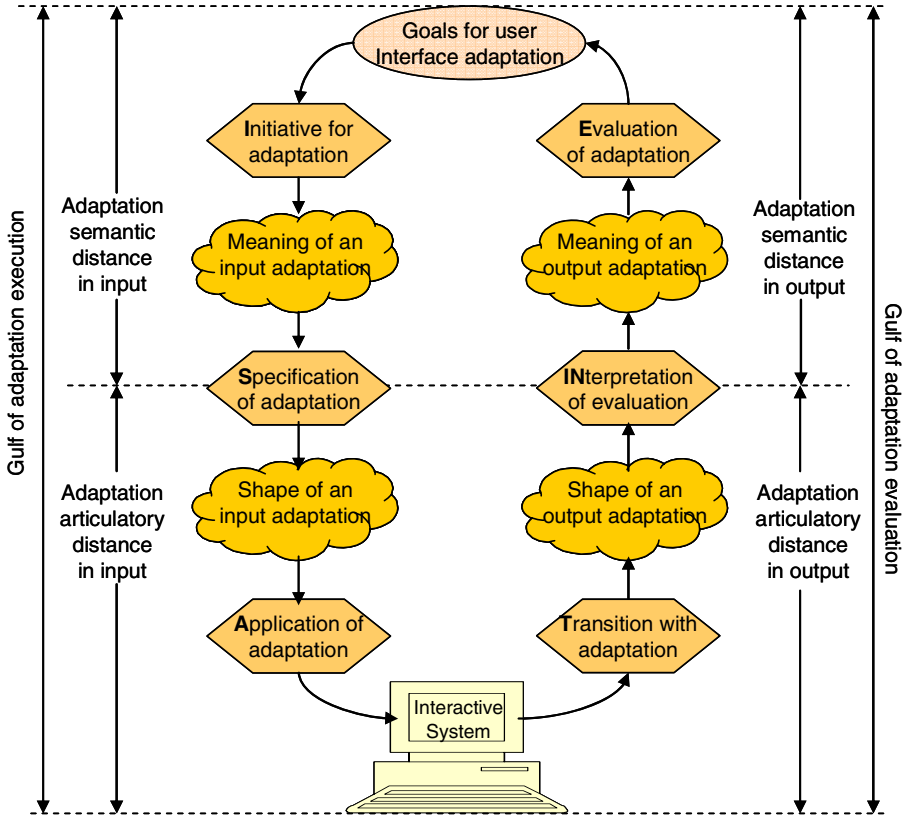

Fig. 2. The seven stages of the Isatine framework for user interface adaptation

3 A Multi-agent Architecture Supporting ISATINE Framework

The previous section identified some holes in the support of a complete adaptation process, which is also reflected in some lacks of system support for these stages. Indeed, the lack of general techniques, methods and tools for adaptation design produces systems where the support for adaptation is rather inflexible, and the knowledge injected into the adaptation engine is very hard to be reused. In the design of a general technique that supports adaptivity in a flexible manner, where knowledge can be reused and integrated with a user interface design method that provides the required formalism to build UIs in a systematic way, a software architecture is required able to cope with all these requirements. However, this software architecture should be able to decide which adaptation could be applied, when they should be applied, etc. Therefore, a dedicated software architecture is required, where the system is able to make some reasoning and to decide what to do next (which adaptation to apply, if any).

Different reasoning models have been proposed so far: rule based systems, neural networks, Bayesian networks, etc. However, a great interest has appeared for software agents [19] as a means to represent reasoning capabilities in an abstract manner

similar to human reasoning. Most of them use the BDI model (*Beliefs*, *Desires*, *Intentions*) [8,18], which is inspired by human reasoning theories. *Beliefs* represent the view the agent has of itself and the world where it is immersed. *Desires* describe the goals that the agent is trying to achieve. Finally, *Intentions* are the plans the agent is executing to achieve the goals it pursues. Because the designed architecture supporting the ISATINE framework should be able to manage negotiation, delegation, and transferring between the different stakeholders in adaptation process (the user, the system or a third-party) multi-agent systems are especially suitable, since there is already some work done within agents research community regarding how the different agents involved in a multi-agent system collaborate or compete negotiating, delegating or transferring duties. Another advantage found in multi-agent systems is the natural distribution of computation, which supports the integration of the implemented multi-agent system with exiting services easily. Furthermore, software agents have already proved useful in the interaction between the user and the UI in some projects such as [8,18]. Those were our motivations to design an architecture to support the ISATINE framework as a set of agents collaborating in a multi-agent system to achieve the final goal: adaptation. Next, how the different stages of the adaptation process defined in ISATINE framework are carried out by the multi-agent system created will be addressed.

3.1 Goals for User Interface Adaptation

The goals for user interface adaptation express the motivations to initiate an adaptation process. When these goals are in the user's head, our system cannot directly achieve them, however the system supports it by means of the adaptability facilities included. Although, not every user goal can be supported, including support for some of them actually increases user's confidence in the adaptation capabilities of the system. When the goals are kept by the system, they should be expressed in terms of the context of use characteristics considered during the design of the system and the usability criteria to be preserved. Thus, no goal can be stored that makes use of context of use characteristics that the system is not able to either query or store. The goals for adaptation kept by the system are represented in two different components in our system. On the one hand, these goals are partially expressed as part of the adaptation rules that will finally produce the adaptations required to fulfill those goals. On the other hand, they are expressed as a usability trade-off. This usability trade-off specifies relatively the usability criteria that should be preserved while adapting the user interface. For instance, if in the usability trade-off we specify that continuity should be maximized, the system will always choose those adaptations producing a lesser disruption in continuity, unless the user forces the execution of another adaptation. This usability trade-off is expressed by using *I** [18] notation. *I** notation was originally designed to specify system goals in early requirements analysis stage. In section 0 how this trade-off is actually applied is described. The multi-agent system supports also those goals remotely-expressed. In this last case, the new remote goals should be expressed in terms of new adaptation rules that can be plugged into the adaptation engine seamlessly. In section 0 we elaborate more on how these adaptation rules are designed and specified.

3.2 Initiative for Adaptation

In ISATINE multi-agent architecture, the adaptation process can be initiated by either the user, the system or a third-party. The user is allowed to do it by clicking or typing (auditory user interfaces are not supported by now) an option available in every user interface generated by the system. The system can decide that an adaptation is needed by inferring it from the incoming information from the context of use. The agents in charge of detecting context of use changes (*AgentContextPlatform*, *AgentContextEnvironment*, *AgentContextUse* and *AgentDetectContextOfUse*) notice those changes by means of sensors. These sensors can be either software or hardware sensors. Hardware sensors are built or plugged into the hardware platform where the application is running, while software sensors are programmed, and included into the applications supported by the multi-agent system. The designer of the adaptation facilities of every application can define his own software sensors provided that the implementation is compliant with the defined interface for sensors. Most data incoming from sensors in directly linked with a piece of information in the context model, although it is not mandatory. In this architecture, the current task the user is carrying out is also included within the context of use, since it is necessary quite often to guess user needs. To guess the user's current goal, this agent uses the task model created at design time. This task model is a tree where the designer specifies the tasks the user will be able to perform along with some temporal constraints (for instance, a sequential relationship between two tasks). Thus, at any time, taking into account the tree structure and the temporal constraints between the tasks, there will be just a set of possible tasks that the user is allowed to perform through the UI (called enabled tasks set). Therefore, the agent just needs to guess which one among the tasks included in the enabled tasks set is the current task. To help in this problem, the agent uses the usage data collected from interaction, especially taking into account the last components of the UI that have been manipulated and the mapping between the widgets of the user interface and the tasks in the task model.

3.3 Specification of Adaptation

Given an initiated adaptation process it is necessary to decide which adaptation will be applied (if any). Whether the user, the system or a third-party has initiated the adaptation process, *AgentAdaptationProcess* Agent proposes the set of adaptation rules that best fit the current context of use. The specification of the set of available adaptations to choose from is built in different ways. The user can demonstrate how he would like the user interface to be adapted. Currently, the user is allowed to demonstrate the colors for each kind of widget, the sizes of the different types of widgets and some kinds of widgets replacements. The agent supports also the specification of rules by computation, although it is currently constraint to the refinement of rules previously defined. However, the main corpus of adaptation rules is provided by the application designer by defining how the system should react to the different situations arising from the interaction.

3.4 Application of Adaptation

By default regardless on who was the one that started the adaptation the system will automatically choose which application to apply. If it was either the user or a third

party the one who initiated the adaptation the agent will ask the user or the third party which adaptation between the eligible ones he would like to apply. Otherwise, or if the user or the third-party delegate the task of choosing the adaptation the *AgentAdaptationProcess* agent will choose the most appropriate ones, creating a ranking of rules. To make this selection the rules are evaluated by means of a set of metrics. Afterwards, the agent will try to execute the rules starting from the highest one in the ranking. If the application of the rule does not meet the usability trade-off specified in the goals for user interface adaptation that rule will be discarded and the agent will try to apply the next rule in the ranking following the same process as for the first rule in the ranking. This process is made until no rule is left in the ranking list or until the agent finds that a ranking has been reached in the list too low for that rule to be applied. The agent has been designed so it will not apply any adaptation rule it does not find good enough (unless the uses forces its execution). Most of the time is better inaction than applying a rule that is not good enough, producing a degradation of user interface usability and damaging user confidence in the system.

3.5 Transition with Adaptation

Making smoother and clearer the transition between the original user interface and the adapted one is very important to avoid confusing the user, and therefore to avoid degrading the user's confidence in the system. ISATINE architecture has been extended with a new agent to support this stage. Although many different kinds of transitions [15] from the original user interface to the adapted one can be imagined, in our case we are just supporting those being general enough to be applied to many different user interfaces, since our transitions are generated at run-time on-the-fly. In section 0 an example of how this stage in applied and how our architecture was extended to support it is shown.

3.6 Interpretation of Adaptation

One of the issues we found when testing adaptive systems is that sometimes the users were not actually aware that an adaptation had been done, and even what the adaptation was for. The same happens when the user makes an adaptation and the system does not understand why the user wanted to perform that adaptation. To address the first issue transition stage can be used. However, to address the second issue another sub-stage is required to help the user in evaluating what the result of the adaptation was. In this sense, if the user is the one leading the adaptation process, she is allowed to provide a description of what the adaptation was useful for. It allows the system to extract some keywords used to relate this new adaptation with other adaptations stored in adaptation rules pool. On the other hand, if the system leads the adaptation process, it always adds a tooltip to the adapted user interface with a short description of the adaptation made.

3.7 Evaluation of Adaptation

An adaptation quality assessment is essential to any good adaptation process, because it should be adaptive itself. The system assesses the adaptation performed by applying heuristics to evaluate a migration cost [13]. This assessment is made at specification

of adaptation stage to create a ranking with the potential applicable rules. However, it is not enough. Since it is impossible to foresee every combination of factors in the context of use, the system can apply a rule not good enough, or simply it can apply a rule the user dislikes. Thus, in the architecture the user can undo any adaptation applied expressing he did not like it. This feedback from the user is injected into the adaptation evaluation mechanism applying a Bayesian approach where rules can improve or worsen their ranking.

4 Implementing the Multi-agent Architecture

In this section we will show an overview of the technologies used in the implementation of the multi-agent architecture to support ISATINE framework.

For the multi-agent system implementation we have used JACK[2] [2]. JACK is an agent programming language based on BDI paradigm. This language generates Java language code out of a set of templates that is executed within an execution environment supplied with the language. To maximize platform independence we have wrapped the multi-agent java based system within an HTTP server interface.

The HTTP server interface allows any platform capable of networking using TCP/IP protocol to access the ISATINE adaptation engine. This HTTP server has been implemented as a servlet (server side applet) that runs on top of a TOMCAT server.

Internally, the user interface knowledge gathered at design-time, and later at runtime by means of sensors is stored by using the XML-based user interface description language UsiXML[3]. By means of UsiXML we are able to achieve the specification of a user interface in a representation abstract enough to be presented in different platforms. The model in this language, which is closer to the actual user interface the user interacts with, is the concrete model.

The concrete UI model describes a UI in a manner independent from the platform where it will run on (although it is dependent on modality). Therefore, a renderer is needed so the user can visualize the UI. For this purpose, a renderer for the concrete UI level of UsiXML has been written for several languages. Currently, there is basic rendering support for XUL, Java 2, J2ME, and OpenLaszlo[4] languages, what allows us to run the developed adaptive applications in almost every platform.

By now, we have just implemented sensors for collecting interaction usage data from the client platform and the user. Other data, such as environment physical conditions changes are being simulated via an agent called *AgentStimuliGenerator*. This agent is able to process an input XML file containing a specification of events and their timing, so it can simulate the arrival of changes in the context of use from hardware or software not currently available. This is especially useful during adaptation rules design process.

The real adaptation the user interface undergoes as a result of the application of the adaptation rules is specified by using Attributed Graph Grammars [17]. A detailed description of how this approach is used to generate a user interface can be found in [12]. The transformation engine to execute the transformations associated to the

[2] http://www.agent-software.com/shared/products/index.html
[3] http://www.usixml.org
[4] http://www.openlaszlo.org

adaptations uses the API from AGG (Attributed Graph Grammars) tool[5] to perform the transformations. It provides a programming language enabling the specification of graph grammars and a customizable interpreter enabling graph transformations.

Next, a description of the main processes within the implemented multi-agent system will be described.

4.1 Receiving Context Changes from the Sensors and Adapting the UI

When a sensor wants to communicate any change in the context of use it has detected, it opens a communication with a specific URL belonging to the *webAdaptationEngine* servlet. Then the sensor will send the information using the XML format designed for this purpose. This information will be passed to *AgentDispatcherAgent* by the servlet. This agent acts as a mediator between the multi-agent system and the servlet. This agent will detect that it is a context communication act, and it will use its plan *ContextEventGenerator* to send the information to the agent *AgentDetectContextOfUse*. This agent will perform two steps: it processes the XML information received and dispatches each piece of information to the corresponding agent (*AgentContextPlatform*, *AgentContextEnvironment* or *AgentContextUser*). A*gentContextPlatform*, *AgentContextUser* and *AgentContextEnvironment* will update the context model to reflect the changes they have received from *AgentDetectContextOfUse*. Notice that not every piece of information received from *AgentDetectContextOfUse* will produce a change in the context model. The new values received can be equal to the values stored in context model, or the changes in the values might not be significant. When these agents update the context model (represented as agents' beliefs - called *PlatformContextModel*), an event will be generated automatically by the agent's beliefs to indicate to *AgentDetectContextOfUse* that it should throw events of the type *ContextChanged*. These events will be handled by *AgentAdaptationProcess*, which will generate the feasible adaptivity rules (plans) for the new context of use. Finally, a meta-reasoning method will be used to choose the rules to be applied using the adaptation rules selection policy chosen. For the execution of the rules, the agent first gets the up-to-date usiXML version of the running UI to be adapted. Next, it transforms the usiXML specification into a graph representation, and it applies the selected rules using AGG API. Finally, the adapted graph is transformed back to usiXML and the target language at the same time. Thus, the adapted UI is made available to the *AgentDispactcherAgent*, so it can be delivered to the client. This process is illustrated in Fig. 3.

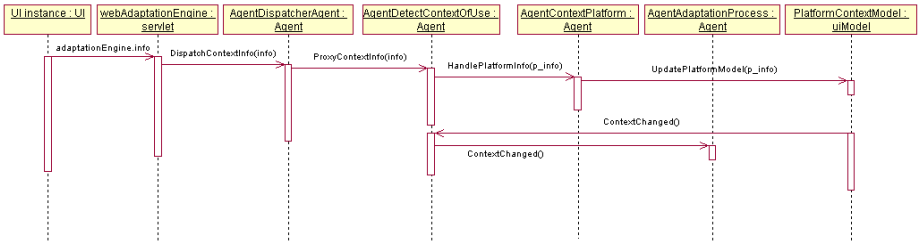

Fig. 3. Receiving Context Changes Info from the Sensors and Adapting the UI

[5] http://tfs.cs.tu-berlin.de/agg/index.html

4.2 Getting the Adapted User Interface

When any of the sensors communicate information to the adaptation engine, they always receive an answer about whether there is a newly adapted UI ready or not. If there is a new adapted UI ready, it will connect to a specific URL belonging to the *webAdaptationEngine* servlet. Then, *AgentDispatcherAgent* will send the adapted UI to the client. Thus, the user will get an adapted version of the UI that matches the changes in the context of use detected by sensors.

5 A Second – Hand Car Selling Case Study

To demonstrate how the architecture supports ISATINE framework for a real example, and the flexibility introduced by designing the architecture as a multi-agent system, a case study is presented next. The case study is based on the main searching facilities form of a real second-hand car selling website. In this form the user is allowed to select the different data required to filter the kind of second-hand car he is searching for. In this sense, the user can provide the car brands he would like the car to be, the maximum amount of money he is willing to spend or the mechanical and physical characteristics of the car. The examples in this section will be presented in growing

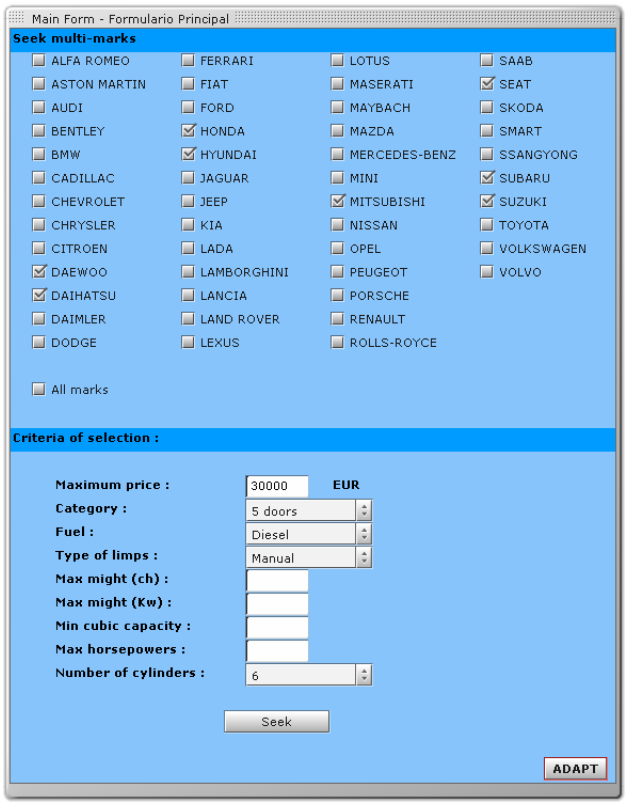

Fig. 4. Original main form for the second-hand car-selling example

complexity to illustrate the features starting from the more simple adaptations to the more complex ones. In Fig. 4, a screenshot of the main form for our example is shown. In this case we have used the OpenLaszlo renderer of our architecture to generate the final running user interface (http://www.usixml.org/index.php?view=page&idpage=120). On the upper part of the form the user selects the car brands he is interested in, while in the bottom part the user selects the features and constraints for the cars he is searching for.

5.1 Adaptability in ISATINE Framework Architecture

One of the main issues in adaptive systems is that if the adaptations are not properly carried out, and the user feels a sense of loosing control, the adaptation engine might be rejected. Therefore, it is really important for an adaptation architecture to support the user in taking control of the adaptation engine, because mental model and tastes for different users might differ. In our example the user is querying the system database for the different car brands he is interested in.

To take this decision he is getting additional information from the web pages of different branches. However, the user in his current context of use is a little bit annoyed with the way the interaction is made, because the form takes too much screen display space. By occupying so much space the form is preventing the user from browsing the car brands web sites while selecting the different car features, forcing the user to switch between the second hand car selling application and the car's websites. At his moment has a goal on her mind: reducing the displaying space required to interact with the application.

Because of that goal the user wants to adapt the user interface to reduce screen space required by the form of the second hand selling application to be shown. To do so, the user clicks on the "*ADAPT*" button to express her intention to adapt the user interface. Next, according to the ISATINE framework, the adaptation to be performed needs to be specified. In this case, in order to specify which adaptation is executed, the user selects the adaptation from the available adaptation rule pool. An adaptation rule replaces a set of checkboxes with a multi-select combo box. In this selection activity, the user is supported by providing a meaningful description of the results achieved by applying the adaptation. The application of the adaptation is made by the system. Since it is the user the one who chose to apply the adaptation it will be applied regardless of the ranking of the rule. Because it was the user who led the adaptation process, it is not necessary to help him to interpret the adaptation. The adaptation in this case is considered to be successful unless the user undoes it.

5.2 Platform Adaptation in ISATINE Framework Architecture

In the same manner as for the user-initiated adaptation previously described, the architecture supports platform adaptation. In this second example, the user is now using the second hand car selling application in a PDA. In this case, the adaptation is triggered as a result of a goal specified by the designer: "every form displayed in the target platform must show, or at least allow browsing, the data required to carry out the task the form is intended to". The initiative in this case is taken by the system. The system detects a change in the target architecture by means of software sensors

reporting the new characteristics of the platform. To face this situation the system uses the set of rules created by the designers. To reinforce user's trust in the system it shows to the user the possibilities to achieve this platform shifting. In our example, the user selects the application of the same rule as in the previous example, so the checkbox group is replaced with a multi-select combo box. In general, one could imagine to provide the user with different sets of rules applicable for each specific platform.

5.3 Context Adaptation in ISATINE Framework Architecture

The user is now at a motor show where many different brands are available. The user is using the application in a PDA equipped with a web cam. The user is making a videoconference to decide which car to buy. So the user stands on the center of the exhibition center and he would like to show to other person each car in the conference, and then by using the second hand car selling application find out if there are any of those cars available and what its characteristics and price are. The user takes the initiative by clicking on the "*ADAPT*" button. The system now offers to the user the list of possible adaptations to apply. In this case, the user chooses an adaptation called "minimum presentation" that transforms the searching form of the application into a minimal set of widgets to allow querying the site for second hand cars. The adaptation application stage is made in this case in collaboration between the user and the system. The system applies the adaptation to produce a minimal presentation, however, it is the user in charge of positioning the brand new generated presentation in the best place of the screen to support his activities.

5.4 Extending ISATINE Framework Architecture to Support Transition Stage

In the previous example, there is an abrupt change between the original user interface in Fig. 5 and the adapted user interface shown in Fig. 6. Thus, a big disruption appears in the change from the original user interface to the adapted one, drastically reducing continuity usability property.

To improve the continuity in the adaptation process a new stage should be included in the adaptation process in charge of making smoother the transition from the original user interface to the adapted one. This stage is one of the extra stages in ISATINE adaptation framework with respect to Dieterich's one. One of the key features of the designed architecture is its extensibility. As long as it was created by using agent's paradigm, it can be easily extended by just adding some extra new agents and rerouting some messages from some agents to other agents. For instance, for the transition state we added a new agent called *AgentTransition*. *AgentAdaptationProcess* was modified so as to send the adapted user interface generated during the application of the adaptation rules to this brand new agent, instead of sending it directly to *AgentDispatcherAgent* to be delivered to the user. *AgentTransition* takes the adapted user interface and it creates smooth transitions depending on the kind of adaptation the user interface has undergone. Right now, this agent is able to highlight the adapted widgets in different ways to guide the user, by changing the background color of some components, changing the panel containing the adapted components or adding word balloons to explain the user what happen during the adaptation. Other techniques such as image animation or morphing could be implemented also. The new adapted user

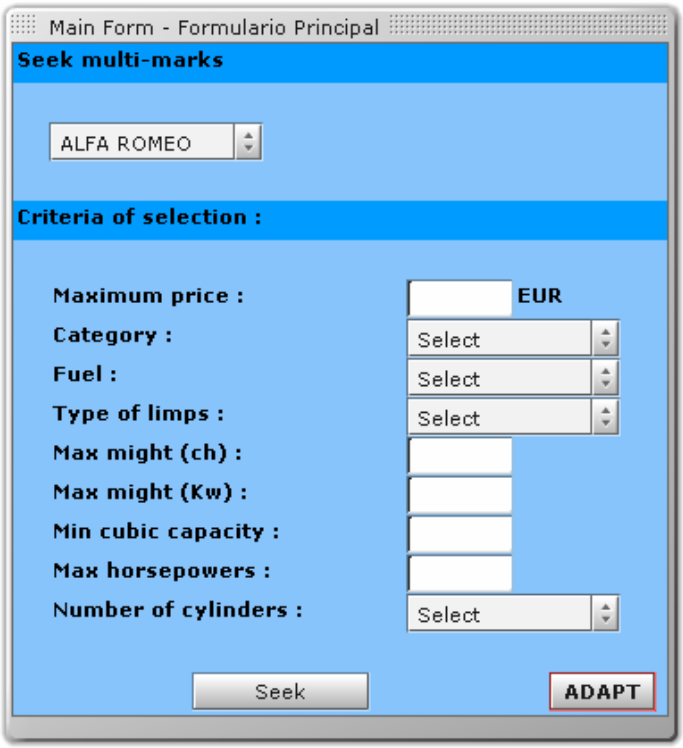

Fig. 5. Adapted user interface reducing displaying space

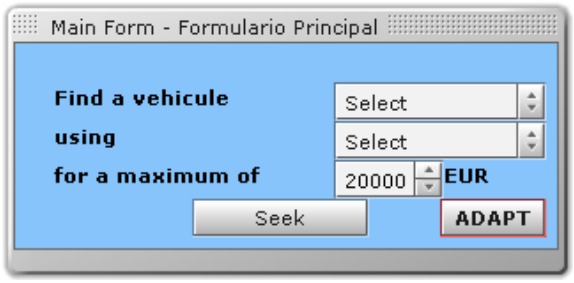

Fig. 6. Second-hand car selling user interface after a context adaptation

interface with the transition effects added is sent to *AgentDispatcherAgent* to be finally delivered to the user. In Fig. 7 a screenshot of the application of the transition stage by *AgentTransition* can be found. A tooltip has been added by *AgentTransition* to remind the user that he can change the view to show some more extra filtering options by clicking on "*ADAPT*" button. In the same manner that *AgentTransition* agent has been added, other extra agents could be added almost seamlessly to extend the architecture to better attend adaptation requirements.

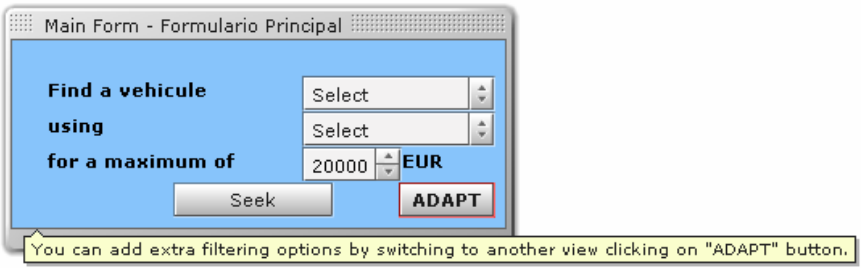

Fig. 7. An example of the output for the transition stage applied to the UI in Fig. 6

6 Conclusion and Future Work

This paper was initially motivated by the need for supporting more than just the adaptation execution, which is addressed in Dieterich's taxonomy. This taxonomy has therefore been expanded according to the Seven Stages of Norman's theory of action, thus, leading to the ISATINE framework for UI adaptation. This framework not only decomposes the whole adaptation process into seven corresponding stages, but it also shows how to decompose each stage into sub-stages depending on the collaboration between the entities involved in each stage: the user, the system, an external third party or any collaboration between them. A multi-agent software architecture has been motivated, justified, and defined so as to support the stages of the framework defined. The BDI paradigm has been used for this purpose. A graph transformation system, consisting of steps of graph transformations, has been developed to support the execution of the adaptation on a UI model. A running example has demonstrated how this architecture should be modified in order to accommodate a series of progressively more complex adaptation schemes, thus validating the approach.

A first area for future work consists in exploring other forms of collaboration such as competition (where at least two entities should compete to find out the best solution and a judge entity then keeps the best one assessed according to some criteria) or coopetition (where at least two entities should compete while cooperating at the same time because their knowledge is perhaps complementary). Coopetition is the combination of cooperation and competition. These new forms do not disrupt the multi-agent architecture defined in this paper. A new agent could be incorporated and new relationships defined according to the BDI paradigm could be defined. This greatly simplifies updating the software architecture for accommodating new forms of adaptation, even perhaps the unknown ones.

A second area for future work is to pursue research and development for the agent responsible for conducting the transition. Many techniques proposed in [15] are very promising for this purpose, but they are built-in. The advantage of having the UI model maintained at adaptation time enables us to develop some of these techniques specialized for the UI widgets.

A third area for future work consist of examining how IFIP quality properties (e.g., honesty, observability, browsability [8]) could be preserved by applying this or that adaptation technique and how controllability and traceability of the stages (especially transition and evaluation) could be achieved.

Acknowledgements

This work is partly supported the Spanish CICYT TIN2004-08000-C03-01 grant and the PBC-03-003 and PAI06-0093-8836 grants from the Junta de Comunidades de Castilla-La Mancha. Also, we gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating HCI similar to human-human communication of the Sixth Framework Program.

References

1. Bratman, M.E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)
2. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents - components for intelligent agents in java. AgentLink News Letter (January 1999) White paper accessible, `http://www.agent-software.com`
3. Calvary, G., Coutaz, J., Thevenin, D.: Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism. In: Blandford, A., Vanderdonckt, J., Gray, P. (eds.) Interactions sans frontières – Interactions without frontiers, Proc. of the Joint AFIHM-BCS Conf. on Human-Computer Interaction IHM-HCI 2001, Lille, 10-14 September 2001, vol. I, pp. 349–363. Springer, London (2001)
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3), 289–308 (2003)
5. Dieterich, H., Malinowski, U., Kühme, T., Schneider-Hufschmidt, M.: State of the Art in Adaptive User Interfaces. In: Schneider-Hufschmidt, M., Khüme, T., Malinowski, U. (eds.) Adaptive User Interfaces: Principle and Practice. North Holland, Amsterdam (1993)
6. Eisenstein, J., Puerta, A.: Adaptation in Automated User-Interface Design. In: Proc. of ACM Conf. on Intelligent User Interfaces IUI 2000, New Orleans, 9-12 January 2000, pp. 74–81. ACM Press, New York (2000)
7. Florins, M., Vanderdonckt, J.: Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In: Proc. of ACM Conf. on Intelligent User Interfaces IUI 2004, Funchal, 13-16 January 2004, pp. 140–147. ACM Press, New York (2004)
8. Gram, C., Cockton, G.: Design Principles for Interactive Software. Chapman & Hall, London (1996)
9. Kolp, M., Giorgini, P., Mylopoulos, J.: An Organizational Perspective on Multi-agent Architectures. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 128–140. Springer, Heidelberg (2002)
10. Horvitz, E.: Principles of Mixed-Initiative User Interfaces. In: Proc. of ACM Conf. on Human Factors in Computing Systems CHI 1999, Pittsburgh, 15-20 May 1999, pp. 159–166. ACM Press, New York (1999)
11. Langley, P.: User Modeling in Adaptive Interfaces. In: Kay, J. (ed.) Proc. of the 7^{th} Int. Conf. on User Modeling UM 1999, pp. 367–371. Springer, Berlin (1999)
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: USIXML: A language supporting multi-path development of user interfaces. In: Bastide, R., Palanque, P., Roth, J. (eds.) DSV-IS 2004 and EHCI 2004. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005)

13. López-Jaquero, V.: Adaptive User Interfaces Based on Models and Software Agents, Ph.D. thesis, University of Castilla-La Mancha, Albacete, Spain (in Spanish) (October 14, 2005), http://www.isys.ucl.ac.be/bchi/publications/Ph.D.Theses/Lopez-PhD2005.pdf
14. Norman, D.A.: Cognitive Engineering. In: Norman, D.A., Draper, S.W. (eds.) User Centered System Design, pp. 31–61. Lawrence Erlbaum Associates, Hillsdale (1986)
15. Rogers, S., Iba, W.: Adaptive User Interfaces: Papers from the 2000 AAAI Symposium, Technical Report SS-00-01. AAAI Press, Menlo Park (March 2000)
16. Schlienger, C., Dragicevic, P., Ollagnon, C., Chatty, S.: Les transitions visuelles différenciées: principes et applications. In: Proc. of the 18th Int. Conf. on Association Francophone d'Interaction Homme-Machine IHM 2006, Montreal, 18-21 April 2006. ACM Int. Conf. Proc. Series, vol. 133, pp. 59–66. ACM Press, New York (2006)
17. Taentzer, G.: AGG: A graph transformation environment for modeling and validation of software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–453. Springer, Heidelberg (2004)
18. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering RE 1997, Washington, 6-8 January 1997, pp. 226–235. IEEE Computer Society Press, Los Alamitos (1997)
19. Wooldridge, M., Jennings, N.R.: Agent Theories, Architectures, and Languages: A Survey. In: Proc. of ECAI-Workshop on Agent Theories, Architectures and Languages, Amsterdam, pp. 1–32 (1994)

Questions

Philippe Palanque:
Question: According to the fact that you are using a multi-agent technology (that is by definition continuously evolving), how can you assess the results and, for instance, guarantee that the adaptation that was a success once, will be a success again?

Answer: this is a real problem and the definition of metrics on a multi-agent platform is still a research topic. Now that the platform is ready and that the architecture is defined this is one of the things we will be working on.